Real-Time Business Intelligence in the MIRABEL Smart Grid System

Fischer, Ulrike; Kaulakiene, Dalia; Khalefa, Mohamed; Lehner, Wolfgang; Pedersen, Torben Bach; Siksnys, Laurynas; Thomsen, Christian

Published in:
Enabling Real-Time Business Intelligence

DOI (link to publication from Publisher):
10.1007/978-3-642-39872-8_1

Publication date:
2013

Document Version
Early version, also known as pre-print

Real-time Business Intelligence in the MIRABEL Smart Grid System

Ulrike Fischer[1], Dalia Kaulakienė[2], Mohamed E. Khalefa[2], Wolfgang Lehner[1], Torben Bach Pedersen[2], Laurynas Šikšnys[2] *, and Christian Thomsen[2]

[1] Dresden University of Technology, Database Technology Group, Germany
{ulrike.fischer,wolfgang.lehner}@tu-dresden.de
[2] Aalborg University, Center for Data-Intensive Systems, Denmark
{daliak,mohamed,tbp,siksnys,chr}@cs.aau.dk

Abstract. The so-called smart grid is emerging in the energy domain as a solution to provide a stable, efficient and sustainable energy supply accommodating ever growing amounts of renewable energy like wind and solar in the energy production. Smart grid systems are highly distributed, manage large amounts of energy related data, and must be able to react rapidly (but intelligently) when conditions change, leading to substantial real-time business intelligence challenges. This paper discusses these challenges and presents data management solutions in the European smart grid project *MIRABEL*. These solutions include real-time time series forecasting, real-time aggregation of the flexibilities in energy supply and demand, managing subscriptions for forecasted and flexibility data, efficient storage of time series and flexibilities, and real-time analytical query processing spanning past and future (forecasted) data. Experimental studies show that the proposed solutions support important real-time business intelligence tasks in a smart grid system.

Key words: BI over streaming data, real-time decision support, tuning and management of the real-time data warehouse, smart grids, renewable energy, flexibility defining data, forecasting

1 Introduction

Production from renewable energy sources (RES) such as solar panels and wind turbines highly depends on weather conditions, and thus cannot be planned weeks ahead. The EU FP7 project *MIRABEL* (Micro-Request Based Aggregation, Forecasting and Scheduling of Energy Demand, Supply and Distribution) [1] addresses this challenge by proposing a "data-driven" solution for balancing demand and supply utilizing *flexibilities* in time and in energy amount of consumption and production.

Figure 1 illustrates energy loads in the electricity grid before (left side of the figure) and after (right side of the figure) the *MIRABEL* solution balances energy in the grid. The solid gray area depicts non-flexible demand (e.g., usage of

* This work is partially done while visiting Dresden University of Technology

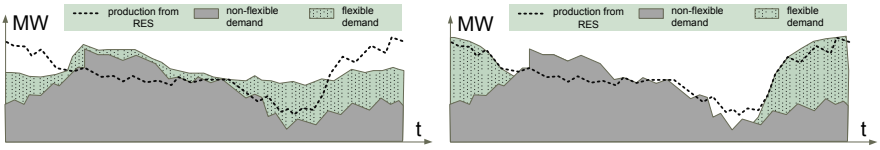
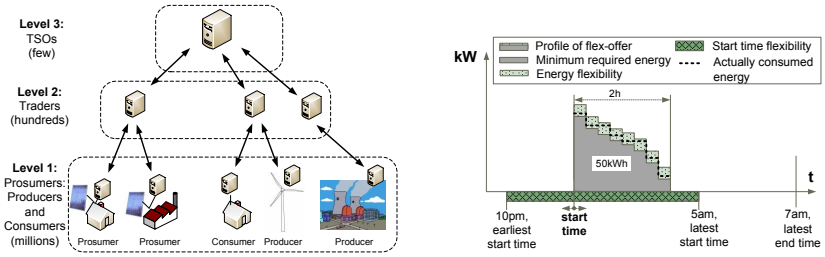


Fig. 1. Balancing consumption and RES production

cooking stoves, lamps, TVs) aggregated from many consumers and varying over some time interval. The dotted area depicts aggregated flexible demand (e.g., usage of washing machines, charging of electric vehicles) that can potentially be shifted to a time when surplus production from RES is available. The dashed line shows available production from RES. The *MIRABEL* project aims to enable the utilization of demand flexibility in time so that higher amounts of the RES production can be consumed (see the right side of Figure 1).

The *MIRABEL* project designs and prototypes an *electricity data management system* (shortly EDMS) that consists of many distributed nodes. These nodes will eventually be deployed at the sites of different actors of the European Electricity Market [2] and their deployment architecture will reflect the hierarchical organization of the European market (Figure 2(a)). Nodes at the lowest level of the system will belong to consumers, producers, and prosumers (entities that both produce and consume electricity), e.g., households, small and large industries. Nodes at the second level will belong to traders, e.g., utility companies or balance responsible parties (BRPs). Finally, the nodes at the highest level will be managed by transmission system operators (TSOs).

The whole EDMS will operate by exchanging and manipulating up-to-date energy related data: (1) time series – measurements of energy consumption and production at each fine-grained time interval, and (2) *flex-offers* – special energy planning objects representing a prosumer's intention to consume (or produce) a certain amount of energy in a specified future flexible time interval. Figure 2(b) shows a flex-offer's energy profile with minimum (solid gray area) and maxi-



(a) Intended EDMS architecture (b) Flex-offer example

Fig. 2. The intended EDMS architecture and flex-offer for charging a car's battery

mum (dotted area) required energy at particular time slots, and the starting time flexibility (shaded area). A flex-offer can also represent demand (or supply) flexibility of a group of prosumers.

In order to meet the project's goals, it is crucial that the EDMS (with all its nodes) is able to efficiently propagate such type of energy data up and down in the hierarchy. Higher level (TSOs, BRPs) nodes must allow storing, querying, forecasting and updating of such data in a timely manner, while lower level (prosumer) nodes might support only the subset of the functionality. In this respect, the *MIRABEL* system is unique due to its real-time requirements, very distributed and hierarchical nature, handling of flex-offers, and the combination of past and future (i.e., predicted) values in querying.

This paper focuses on the data management aspects of a higher level (BRP or TSO) node while leaving those aspect associated with the distribution of nodes for future work. First, it exemplifies the real-time aspects encountered in the EDMS by providing a real-world *MIRABEL* use case. Then, it presents the initial work done designing and implementing a *real-time data warehouse* that operates locally in a node and offers near real-time data management and querying of the two essential entities in *MIRABEL* – time series and flex-offers. The paper thus presents 1) the architecture of the data warehouse, 2) underlying real-time business intelligence challenges, 3) internal warehouse components, and 4) initial results of experiments on individual components.

The rest of the paper is organized as follows. Section 2 presents the use case example of the *MIRABEL* system. Section 3 introduces the data warehouse architecture and the data flow between internal components. The individual components and their underlying real-time challenges are presented in Sections 4–8. Initial experimental results for some components are presented in Section 9, followed by related work in Section 10 and the conclusions and future work in Section 11.

2 MIRABEL Use-Case

In order to demonstrate the typical flow of events in the *MIRABEL* system, we employ the example of charging an electric vehicle utilizing the system.

1. A user of an electric car comes home at 10pm and wants to charge the battery of the car before next morning for the lowest possible price. Once plugged in, the outlet recognizes the car and chooses a default energy consumption profile, according to which the totally required energy is 50kWh and the default charging completion time is 7am.
2. The consumer's node of the EDMS automatically generates a flex-offer (see Figure 2(b)) and sends it to the trader's node of the EDMS where it is aggregated with other similar flex-offers and then scheduled. While taking into account external weather and locally computed energy production and consumption forecasts, the trader's node schedules this particular flex-offer as a part of a larger (aggregated) flex-offer so that charging starts at 1am. This

lowers the energy demand peak at 11pm and consumes surplus production of RES (e.g., wind) at 1am. The trader sends to the consumer node a schedule satisfying the original flex-offer.
3. Simultaneously, the trader's node collects streams of energy measurements to be able to forecast demand and supply in the future. When newly arrived measurements differ substantially from current forecasted values, the respective forecasting model parameters are updated transparently.
4. At 12pm, the TSO notices an upcoming shortfall of supply at 1am–2am and instructs the trader to reduce its demand by a specific amount during this period. The trader, consequently, reschedules the consumer's flex-offer to start energy consumption at 3am and sends an updated schedule back to the consumer's node, also with an updated price of the energy.
5. The consumer;s node starts charging the battery of the electric vehicle at 3am and finishes the charging at 5am.
6. Next month, the trader sends to the consumer an energy bill which reflects the reduced energy costs for user's offered flexibility.

In *MIRABEL*, each flex-offer has two attributes (*assignment-before* and *start-time*) which indicate the latest possible time a flex-offer is allowed to be scheduled and the latest possible time the delivery of the energy has to be started, respectively. These timing attributes of a flex-offer impose that critical real-time constraints must be respected by the EDMS. Consequently, the forecasting, aggregation, and scheduling[1] must start as soon as new data is available and complete before the deadlines imposed by the flex-offer timing attributes values. This is not trivial as a node (at trader or TSO level) must deal with millions of flex-offers (expected number) collected from a few hundred thousands of prosumers. In addition, for a longer term energy planning (day-ahead, week-ahead) or risk analysis, a support for advanced analytics over such volatile data is required. As a consequence, a node of the EDMS must be equipped with an efficient data warehouse to be able to process and analyze queries on energy data in near real-time. These queries can also be on past or future data. To meet these requirements in real-time, our proposed approach stores the underlying time series as models (see Section 4). In the next section, we present the architecture of such an energy data warehouse. Note, we use the term real-time when referring to the "soft" real-time or the near real-time concept.

3 System Architecture

The *MIRABEL* EDMS consists of distributed, non-centralized (although hierarchical) nodes with homogeneous communication interface. In this section, we present the architecture of the single EDMS node and focus on our envisioned energy data warehouse which is based on PostgreSQL.

[1] The reliability of nodes and the latency of communication is not considered in this paper.

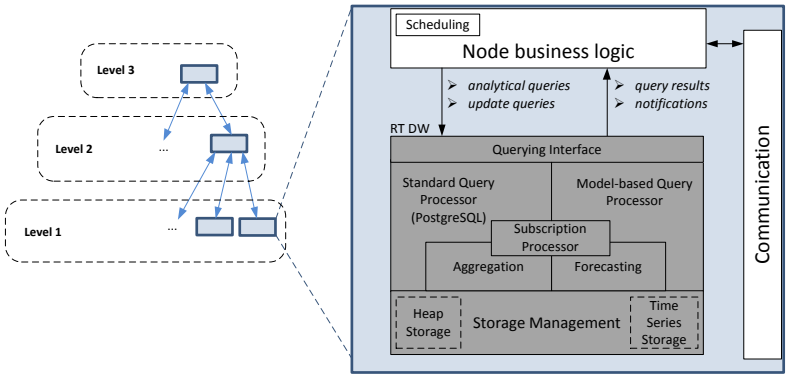


Fig. 3. Internal node architecture

The *MIRABEL* use-case leads to multiple preliminaries and requirements for the needed warehouse. First, time series and flex-offer data storage and querying need to be supported inherently (among other types of energy data). Large volumes of updates of such volatile data must be efficiently processed. Second, forecasting of time series values must be performed efficiently and the querying of such forecasted values must be supported. Finally, advanced queries offering different processing times and accuracy of results must be enabled. Such queries are needed in *MIRABEL* as various data processing tasks require different precision and different access times of data, e.g., invoicing requires precise and detailed data, but there are no constraints on the processing time (i.e., no near real-time processing is involved). Real-time monitoring, on the other hand, requires fast access time, but tolerate approximate or aggregated results. Thus, queries should provide different results – approximate (trend-analysis, long term energy planning), with some imprecision (grid load balancing), and very precise (billing, risk analysis). In the following paragraphs, we present the *real-time data warehouse* and show how it handles such energy data and queries.

On the right side of the Figure 3, we show the EDMS node architecture. It is composed of a Communication component, the Node Business Logic, and a *real-time data warehouse* (RT DW). In this paper, we discuss issues of the *real-time data warehouse* (depicted in gray). It is a layered system consisting of seven individual components, namely, *Storage Management*, *Aggregation*, *Forecasting*, *Subscription Processor*, *Standard Query Processor*, *Model-based Query Processor*, and *Querying Interface*. The *real-time data warehouse* receives analytical and update queries and provides query results as well as notifications to the node business logic. Part of this business logic is the *Scheduling* component (see more details in Tušar et. al. [3]), which implements a particular flex-offer scheduling strategy depending on the business goals of the market player (owner of the node). The node business logic communicates with the other nodes to receive and propagate data such as measurements, flex-offers, or schedules.

Storage Management provides efficient storage for energy data including flex-offers and time series, such as weather data as well as consumption and production measurements. This component is equipped with an intermediate main-memory data store, which accumulates arriving data, makes it available for queries, and materializes it later on (in standard PostgreSQL *Heap Storage*). Additionally, *Storage Management* stores time series models and their parameters (in *Time Series Storage*). *Aggregation* is responsible for incrementally and efficiently aggregating multiple flex-offers into fewer ones with larger energy amount values. Based on time series values, *Forecasting* builds and maintains forecast models and makes future values of time series available for queries. *Subscription Processor* monitors *Aggregation* and *Forecasting* and notifies subscribers when aggregated flex-offers or forecast values change substantially (compared to previous notifications). There are two query processors employed by our data warehouse: *Standard Query Processor* and *Model-based Query Processor*. *Standard Query Processor* (PostgreSQL) accesses *Heap Storage* and provides exact results for user-issued queries. Such queries might be complex and long-running and might involve aggregated and non-aggregated data such as flex-offers and time series. *Model-based Query Processor* relies on historical and future time series models (stored in *Time Series Storage*) and allows approximate but efficient answering of historical, future, and combined queries. Finally, *Querying Interface* offers a unified interface for queries and query results.

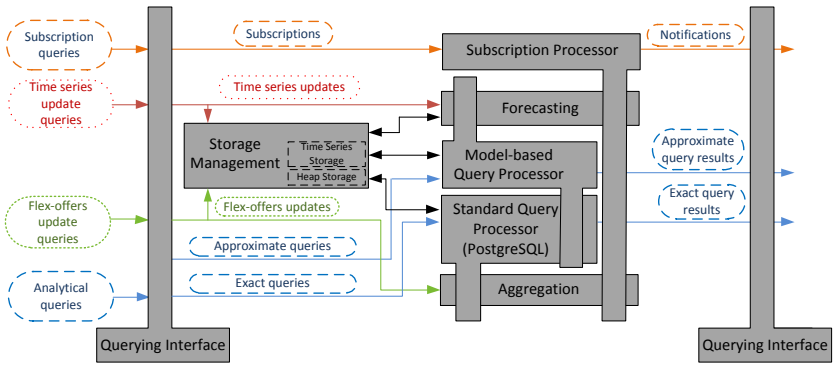


Fig. 4. Data flow during query processing in the real-time data warehouse

Figure 4 visualizes the flow of the queries and the data during query processing in the *real-time data warehouse*. Each query type is first processed by *Querying Interface* and then handed to the appropriate components. The flow path of queries and data differs depending on the type of queries being executed in the data warehouse:

Subscription queries are handled by *Subscription Processor*. Such queries allow users to (un-)subscribe (from) to notifications that are generated every

time results of a user-specified continuous query changes more than a specified threshold. Two important types of continuous queries (among others) are supported by our data warehouse: flex-offer aggregation and time series forecasting queries. One example of a subscriber is the *Scheduling* component which is only informed when forecasts or aggregated flex-offers change substantially.

Time series update queries (value inserts and deletes) are passed to *Storage Management* and *Forecasting* simultaneously. The *Storage Management* component handles the materialization of these updates, while the *Forecasting* component uses these updates to incrementally maintain existing forecasting models. Those models are monitored by *Subscription Processor*.

Flex-offer update queries are processed similarly to the time series update queries and passed to *Storage Management* and *Aggregation* simultaneously. If aggregated flex-offers change substantially, a notification is generated by *Subscription Processor*.

Analytical queries can be either exact or approximate queries. Exact queries are routed to *Standard Query Processor*, while approximate queries are passed to *Model-based Query Processor*. While processing exact queries, the *Standard Query Processor* component accesses *Heap Storage* and can take advantage of *Aggregation* to answer analytical queries over flex-offers. Similarly, *Model-based Query Processor* can utilize *Forecasting* to answer queries on future (or both historical and future) data from *Time Series Storage* and utilize *Standard Query Processor* to answer an exact part of the queries.

In the following sections, we discuss individual components of *real-time data warehouse* in more detail.

4 Storage of Energy Related Data

In this section, we focus on the *Storage Management* component and discuss flex-offer and time series storage as well as data loading.

4.1 Flex-offer Storage

To store flex-offer data, we employ a multi-dimensional schema that includes several *fact* and *dimension* tables (more details about the schema can be found in Šikšnys et. al. [4]). The dimension tables represent time intervals, metering points, possible states of flex-offers as well as legal entities. The fact tables represent flex-offers themselves. The schema is designed to enable convenient storage of energy demand and supply flexibilities and efficient processing of the most common analytical queries in the *MIRABEL* system.

Essentially, there are two fact tables called *F_flexOffer* and *F_enProfileInterval*. They hold flex-offer facts and information about energy profiles (composed of multiple intervals) of a flex-offer, respectively. Each stored flex-offer is given a unique identifier. For each stored profile interval, there are measures to specify the duration of the profile interval as well as the lowest and highest amount

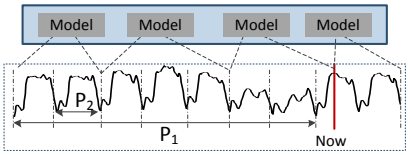


Fig. 5. TimeSeries with associated models

of energy needed. The *F_flexOffer* table holds information about time deadlines associated with flex-offer, i.e., *enProfile_startAfter* time for initial or *enProfile_startFix* time for scheduled flex-offer. As flex-offers can be aggregated into larger flex-offers, we also have the table *F_aggregationMeta* which references *F_flexOffer* twice to point to the aggregated "parent flex-offer" and the smaller "child flex-offer" which has been aggregated, respectively.

Our proposed schema allows to read and write flex-offers efficiently, which is important for flex-offer aggregation as well as other analytical queries, e.g., those used in the energy balancing.

4.2 Time Series Storage

We employ different time series storage schemes which are utilized in the processing of approximate queries with allowed impressions and exact queries with the precise result.

For exact queries, we utilize standard PostgreSQL *Heap Storage* to store the original time series as a block-based array. The benefits of storing the time series as a block-based array are as follows: (1) it alleviates the need for sorting the time series, e.g., when optimizing forecast model parameters, (2) it eases the access to values as only relevant pages (blocks) with requested values have be accessed when answering a query. To determine the range of needed pages, the WHERE clause of a query can be utilized.

For approximate queries, we compactly represent time series over past and future using models and put them in the *Time Series Storage*. To achieve this, we build a set of models that incorporate seasonal and non-seasonal behavior. As time proceeds, the system incrementally updates the models and utilizes them to answer approximate queries. To find the suitable set of models, we divide the time series into non-overlapping sub-intervals (e.g., using approaches in [5, 6]). To build a model over the interval, we decompose the portion of time series over this interval into *trend*, *seasonal*, and *remainder* components utilizing the information about the seasonal periods. A time series can have an arbitrary number of seasonality periods (e.g., the time series shown in Figure 5 exhibits the two seasonality periods P_1 and P_2). Each model represents a segment of the underlying time series with the predefined error guarantee (top of Figure 5). For query optimization, we store statistics about the index in the system catalog.

Specifically, we store the number of models segments and their average size. These statistics are needed to estimate the expected cost for the approximate query.

As explained in our previous work [7], models built for each portion of the time series can also be composed into a hierarchical structure, where models at different levels represent a time series with a certain (bounded) precision. To summarize, time series are stored in the data warehouse as block-based arrays and also as models that represent parts of time series at various precisions.

4.3 Data Bulk Loading

Flex-offers as well as values of time series are initially cached before they are physically stored on disk. First, arriving data is recorded in an intermediate data store in main memory similar to RiTE [8] such that the data is made available for querying before it gets materialized on the disk. Finally, the data is physically inserted into the underlying data warehouse in bulks.

The storage of the data, might, however, still be slow due to high update rate (that might occur sporadically). Instead, for time series, measured values can be ignored if forecast models predict the value within the bounded imprecision. If not, the value can be used to update the forecast model while storing only new model parameters in the data warehouse. Since only models are stored, there are less data values to store and the processing is much faster.

The main advantage of the latter approach is that it supports fast (although imprecise) data loading that enables approximate answering of analytical queries. It also allows transparent processing both of queries that need only fast and approximate answers and of queries that need more accurate answers since all the data can be made available in forecast models. Further, the model-based storage can lead to compression of the data which in turn means that the data may fit in main memory.

5 Real-time Flex-offer Aggregation

Flex-offer aggregation is triggered when a consumer sends a new flex-offer or trader wants to reschedule older flex-offers (see Section 2). The *Aggregation* component combines a set of many *micro* flex-offers into a set of fewer *macro* flex-offers. It is a complex operation as, even for two flex-offers, there are many different ways to combine them into one flex-offer. Each of these combinations results in different flexibility loss, which often needs to be minimized. Fortunately, by grouping flex-offers carefully, it is possible to aggregate many flex-offers efficiently (in linear time) while still preserving lots of their flexibilities [9]. As presented in Section 2, a very large number of flex-offers need be scheduled in *MIRABEL*. Since scheduling is an NP-complete problem [10], it is infeasible to schedule all these flex-offers individually within the (short) available time. Instead, flex-offers should be aggregated first, then scheduled, and finally *disaggregated*. In this process, the additional disaggregation operation transforms

macro scheduled flex-offers into *micro* scheduled flex-offers. As the aggregation plays a major role during the scheduling, we focus on real-time aspects in this particular use-case of the aggregation in the following sub-sections.

5.1 Real-time Aggregation Challenge

In *MIRABEL*, a higher level (e.g., BRP) node receives a continuous stream of flex-offers from many lower level (e.g., prosumer) nodes. Simultaneously, execution deadlines of the previously received flex-offers are approaching and thus their respective individual schedules (assignments) have to be prepared and distributed back to the original issuers of the flex-offers. When aggregating those flex-offers, a number of requirements have to be satisfied [9, 11]:

Disaggregation requirement Aggregation must ensure that it should be always possible to correctly disaggregate scheduled macro (aggregated) flex-offers into scheduled micro (non-aggregated) flex-offers while matching two schedules produced before and after disaggregation. Here, two schedules match when total energy amounts at every time interval are equal.

Compression and flexibility trade-off requirement Aggregation should allow controlling a trade-off between the number of aggregated flex-offers and the loss of flexibility, i.e., the difference between the total time/amount flexibility before and after the aggregation.

Aggregate constraint requirement It must be possible to bound the total amount defined by every aggregated flex-offer, e.g., in order to meet power grid constraints.

After flex-offers have been aggregated, they have to be scheduled, and disaggregated thus producing individual schedules that can be distributed back to the lower level nodes, i.e., prosumers. In *MIRABEL*, the total time available for the complete process of flex-offer aggregation, scheduling, and disaggregation is 15 minutes. This deadline should not be missed as a new set of flex-offers and new forecasts are expected to be available after this period of time. There is a challenge to build such a flex-offer aggregation solution which would satisfy all above-mentioned requirements as well as be scalable enough to deal with millions of flex-offers while still providing a sufficient amount of time for scheduling and disaggregation (from the totally available 15 minutes interval).

5.2 Real-time Aggregation Solution

Considering frequently changing flex-offers in the *MIRABEL* system, we build an efficient incremental flex-offer aggregation solution [9], which satisfies all the above mentioned requirements and aims to address the real-time challenge.

First, the disaggregation requirement is inherently satisfied by the solution, because the aggregated flex-offer always defines less (or equal) flexibilities compared to the original flex-offer and thus it is always possible to disaggregate scheduled flex-offers. Second, we introduce the set of aggregation parameters,

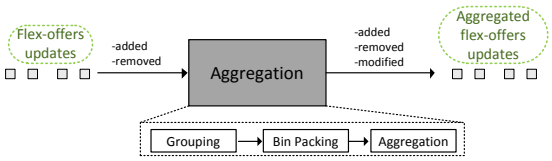


Fig. 6. Input and output of the incremental aggregation solution

and different values of these parameters allow controlling the trade-off between compression and the flexibility loss. Third, we introduce an intermediate *bin-packing* step that ensures that a user-selected constraint, e.g., total maximum amount is below 1000 kWh, is satisfied for every aggregated flex-offer. Finally, our aggregation solution inherently supports incremental aggregation, which provides a huge saving in time when aggregating a flex-offer set that gradually changes in time (as described above). When aggregating flex-offers incrementally, micro and macro flex-offer sets are maintained. As visualized in Figure 6, our incremental solution takes the sequence of micro flex-offer updates as input and produces the sequence of macro (aggregated) flex-offer updates as output. An update in the input contains a flex-offer and indicates whether the flex-offer is *added* or *removed* to/from the micro flex-offer set. Our solution can process such updates one-by-one at a time or in bulks. When the bulk of updates (or a single update) is processed in the *Grouping*, *Bin-packing*, and *Aggregation* steps, the solution updates the set of macro flex-offers and outputs those aggregated flex-offers that either were *added*, *removed*, *modified* during this process. Such incremental behavior is the key addressing the real-time challenge as it allows efficiently processing flex-offers received from lower level nodes as well as those with elapsed execution deadlines. Later in the experimental section, we evaluate the throughput of updates which can be handled by our incremental aggregation solution when one or more updates are processed at a time.

As future work, the support for other types of flexibilities will be provided, e.g., a *duration flexibility*, which allows a BRP to supply a certain amount of energy to prosumers in the preferred duration of time.

6 Real-time Time Series Forecasting

The *Forecasting* component enables predicting the future development of time series data. In *MIRABEL*, forecasts of energy production and consumption time series are crucial in order to enable the scheduling of aggregated flex-offers (see step 3 of the *MIRABEL* use-case in Section 2). Additionally, the *Forecasting* component needs to efficiently process new energy measurements to detect changes in the upcoming energy production or consumption and to enable the rescheduling of flex-offers if necessary.

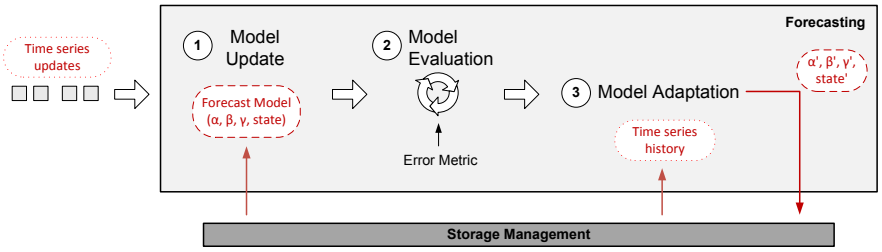


Fig. 7. Overview of forecast model maintenance

6.1 Overview Time Series Forecasting

Sophisticated forecasts require the specification of a stochastic model, a so-called *forecast model*, that captures the dependency of future values on past values. Specific characteristics of energy time series like multi-seasonality (daily, weekly, annual) or dependency on external information like weather or calender events require the employment of forecast models tailor-made for the energy domain [12], [13]. Such models involve a large number of parameters leading to high model creation times due to expensive parameter estimation. Thus, models are pre-computed and stored in the database to support real-time answers.

However, a continuous stream of new measurements of energy production and consumption leads to evolving time series and thus requires continuous maintenance of the associated forecast models (Figure 7) [14]. For each new measure value, we need to initiate (1) model update, which consists of an incremental state adaption of the forecast model. Then, (2) model evaluation checks the forecast model error with regard to the new measurements. Finally, (3) model adaption might be triggered in order to re-estimate the model parameters and, therefore, adapt the forecast model to the changed time series characteristic. Model adaption is as expensive as the initial model creation since most parameters cannot be maintained incrementally.

As the calculation of forecast values from existing models can be done very efficiently, model maintenance exhibits the major challenge in terms of real-time processing. Here, we face two main challenges: (1) when to trigger forecast model maintenance and (2) how to efficiently adapt the forecast model parameters. Both aspects strongly influence the amount of time spent on maintenance as well as the forecast accuracy. In the following, we further detail these main challenges of model maintenance and sketch different solutions that utilize standard as well as energy-domain-specific techniques.

6.2 Real-time Model Adaption

Model adaption might be triggered periodically by a fixed interval, either after a fixed number of new measurements or after a fixed time interval. However, these strategies exhibit the problem of determining the model adaptation interval.

Too short intervals lead to unnecessary adaptations of well-performing forecast models, while too long intervals lead to the problem of unbounded forecast errors in the sense that arbitrarily large errors are possible between the defined points of model adaptation. The other strategy is to utilize the continuous stream of consumption and production measurements to continuously evaluate the forecast model accuracy and trigger model adaptation when the forecast error violates a previously defined threshold. While this strategy guarantees that a certain forecast error is not exceeded, it exhibits a similar drawback as the fixed interval model adaptation since it also depends on the definition of suitable thresholds. To weaken the disadvantages of a single technique, we can combine both evaluation approaches and trigger model adaption time- and error-based.

Model adaption re-estimates the model parameters and therefore, adapts the forecast model to the changed time series characteristics. There, the d parameters of a forecast model span a logical d-dimensional search space, in which we want to find the global optimal solution with regard to the forecast error. In this context, two challenges arise, first, where to start the parameter re-estimation process and, second, how to efficiently re-estimate the parameters itself.

Our core assumption of start point determination is that forecast model parameters will not change abruptly but will be in close neighborhood of previous model parameters. Thus, we should exploit context knowledge from previous model adaption by either using the parameters of the outdated model or a combination of a set of older models. For the last approach, we store previous models of a time series in conjunction with their context information (e.g., weather information) in a repository. If a similar context occurs and maintenance is triggered, we reuse these models to calculate suitable start points [15].

To actually determine the new model parameters, arbitrary global and local optimization methods, each with advantages and disadvantages, can be utilized. Global optimization methods might find a global optimum but need more time to terminate, while local optimization methods follow a directed approach and therefore converge faster to a solution but exhibit the risk of starvation in local sub optima. Again, we can combine both approaches to weaken their disadvantages by starting with local optimization method and using a global optimization method afterwards if there is still time available.

Additionally, we have developed several approaches that utilize characteristics of energy-specific forecast models to speed up the parameter estimation process itself. First of all, energy demand models often involve multi-equation models [13] that create a different forecast models for different time intervals (e.g., for each hour of the day). As multi-equation models consists of several independent individual models, we reduce the parameter estimation time by partitioning the time series and parallelizing the estimation process [16]. Second, energy demand and supply models often require the inclusion of many external sources (e.g., weather information) to increase the accuracy. The naive approach of adding each external time series directly to the forecast model highly increases the model maintenance time due to the large parameter space. Therefore, our

approach separates the modeling of external sources from the actual forecasting model and combines both models after parameter estimation.

Up to now, we have only discussed how to efficiently maintain a forecast model for a single time series. However, we might also exploit the hierarchical organization of the data within one node or over several nodes to reduce the overall number of forecast models and thus the maintenance overhead [17].

7 Subscription Processor for Forecasts and Flex-offers

Within *MIRABEL*, forecasts and aggregated flex-offers are continuously required by scheduling and, thus, might be continuously polled from the data warehouse in each interval. This is very inefficient if flex-offers and/or forecasts changed only marginally leading to high application costs. To prevent the polling, continuous queries can be registered in the data warehouse leading to notifications to subscribers every time when results of these queries change substantially (due to data changes in the warehouse). Such subscriptions and notifications are handled by the *Subscription Processor* component. In the following sections, we provide examples of continuous flex-offer aggregation and forecasting queries and explain how the respective notifications are generated.

7.1 Flex-offer Notifications

Continuous flex-offer aggregation queries must define an error threshold, which define a condition for a notification. For example, the following continuous aggregation query requests aggregated flex-offers for the upcoming day specifying the error threshold of at most $10kWh$:

```
SELECT *
FROM AggregateFlexoffers (
     SELECT flexOfferId FROM F_flexOffer
     WHERE enProfile_startAfter = now() + INTERVAL '24 hours')
NOTIFY ON energy_error > 10000
```

A notification is sent to the subscriber when inserts and deletes of flex-offers cause that the result of aggregated flex-offers changes higher than $10kWh$ in the terms of energy amount. A subscriber can specify whether the notification should include the new set of aggregated flex-offers or only the difference compared to the previously reported aggregation result. The proposed flex-offer notification schema implements data pushing rather than pulling approach, thus it prevents the repeated aggregation of the same set of flex-offers in cases when up-to-date aggregates are required (which is common in the *MIRABEL*). Furthermore, they can lower node's communication costs associated with the propagation of flex-offers throughout the hierarchy of the EDMS.

7.2 Forecast Notifications

Similarly to continuous aggregation queries, subscription-based forecast queries can be utilized. The following continuous forecast query requests forecasts for at least the next 12 hours providing an error threshold of 10%.

```
SELECT hour(time), SUM(energy) energy_per_hour
FROM TS_PowerConsumption GROUP BY hour(time)
FORECAST energy_per_hour ON hour(time)
NOTIFY ON MIN HORIZON '12 hours' AND THRESHOLD 0.1
```

For this query, a notification is sent to the subscriber if either (1) time has exceeded and the subscriber holds forecast values for less than 12 hours (time-based) or (2) if new forecasts are available that deviate by more than 10% from old forecasts sent before (threshold-based). As the subscriber only specifies a minimum number of forecast values, the system can internally decide how many values to send and thus regulate costs. If only a small number of forecast values are propagated, many time-based notifications occur. In contrast, propagating a larger number of forecast values leads to a higher forecast error and thus more threshold-based notifications. Thus, the forecast horizon can either be chosen manually by the subscriber or automatically [18] to ensure accurate forecasts and smaller number of notifications.

8 Query Processing

The real-time data warehouse has two types of query processors called *Standard Query Processor* and *Model-based Query Processor*, respectively. In the following sections, we elaborate the types of queries these two processors support and provide examples of the most important query types in our system.

8.1 Standard Query Processing

Standard Query Processor supports both simple (point, range) and analytical (aggregate, join) queries over historical time series and flex-offer data. All tuples (from *Heap Storage*) relevant for a query have to be retrieved from the main-memory or disk storage in order to process the query, hence processing complex analytical queries over large datasets (of time series or flex-offers) might take a long time. To improve performance, a user can explicitly require the flex-offer aggregation in a query. In this case, *Standard Query Processor* might take advantage of materialized aggregated data (e.g., from *Aggregation*), thus lowering not only data access costs, but also computational costs (e.g., when exploring the flexibility space of flex-offers). In all these cases, the result of a query is always exact. To implement *Standard Query Processor*, a traditional query processor (PostgreSQL) can be utilized and extended with additional user defined functions specific for handling energy data such as flex-offers.

We now give some examples of queries handled by *Standard Query Processor*. The query below shows how the balance between produced and consumed energy at the granularity of 15 minutes can be computed for January of 2012:

```
SELECT hour_quarter(time), SUM(p.energy) AS Prod,
SUM(c.energy) AS Cons, SUM(p.energy - c.energy) as Balance
FROM TS_PowerProduction p, TS_PowerConsumption c
WHERE year(time) = 2012 AND month(time) = 1
GROUP BY hour_quarter(time)
```

The next query finds maximum and minimum energy amounts that potentially can be scheduled at every time interval (of 15 minutes) utilizing all unscheduled flex-offers:

```
SELECT MAX(en_high) as MaxEnergy, MIN(en_low) as MinEnergy
FROM forAllFlexofferTimeShifts(
       aggregateFlexoffers(
          SELECT flexOfferId FROM F_flexOffer
          WHERE enProfile_startFix is not NULL))
GROUP BY timeIntervalId
```

Here, the function *aggregateFlexoffers* aggregates flex-offers with ids specified by a sub-query; for every flex-offer, the function *forAllFlexofferTimeShifts* enumerates its possible alternatives to fix (to schedule) flex-offers in time. The latter function outputs several alternatives containing the minimum (`en_low`) and maximum (`en_high`) energy amounts for each time interval (`timeIntervalId`). Enumeration of such alternatives might be very expensive if aggregation is not performed. Thus, the applied aggregation operation substantially improved the performance of such query. Materialized results of flex-offer aggregation might further improve the performance of such queries.

8.2 Model-based Query Processing

Model-based Query Processor is designed for time series, primarily. It operates on time series models (*Time Series Storage*) rather than tuples. It is usually faster than *Standard Query Processor* as less data needs to be accessed on disk. Furthermore, queries supported by the processor can be executed on both historical and predicted data thus allowing to look into the history and future. To query historical data, a user may explicitly specify error thresholds (absolute error and confidence) that describe how large impressions are allowed to be. Smaller values of the threshold leads to longer query execution times as a larger number of models have to be retrieved from storage (see Section 4.2). If historical models cannot guarantee a required precision, the *Heap Storage* is accessed through *Standard Query Processor*. To query forecasted data, the functionality of *Forecasting* is utilized.

For example, the following query finds the energy consumption of households for the previous 24 hours and upcoming 2 hours with max 5% error:

```
SELECT hour(time), SUM(energy) AS energy_per_hour
FROM TS_PowerConsumption
WHERE category='household' AND
      time BETWEEN (now() - INTERVAL '24 hours') AND
                   (now() + INTERVAL '2 hours')
```

```
GROUP BY hour(time) ORDER BY hour(time)
RELATIVE_ERROR 0.05;
```

Such type of queries focusing on recent history and near future is important in real-time monitoring and intra-day planning of energy. In this process, the timely delivery of the query result is vital.

To summarize, two query processors are supported by the real-time data warehouse. A user might chose to execute his query on either of them, depending on the available time for query execution and the required precision of the result.

9 Experimental Results

We now present some initial experimental results on the storage, aggregation and forecasting components. No other smart grid system does what the *MIRABEL* system does, e.g., with respect to flex-offers, and thus we do not compare with other systems.

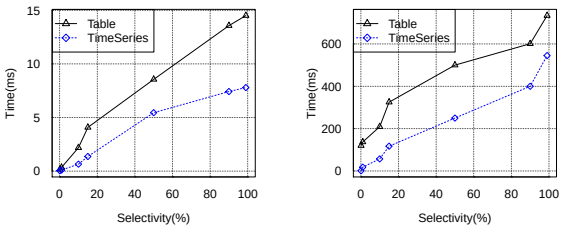
9.1 Storage Experiments

We evaluate query performance for the storage component presented in Section 4.2 using real-world time series dataset of 110,000 tuples. These experiments were run on a computer with Intel CoreI-7 with 8 GB of RAM, and Ubuntu 2.6 (x86 64) OS. In this set of experiments (Figure 8), we compare the performance of representing the time series as a database table (solid line) and natively as a disk-block array (dotted line) by measuring the query time for: (a) a simple query which only accesses the materialized time series and (b) the overall time for accessing the time series and building the forecast model. We increase the selectively of the query from 0.1% to 100% of the dataset.

In Figure 8(a), we achieve 4.5 times speed up for small range queries (i.e., selectively of 0.1%) and up to 1.8 times speed up for almost the entire dataset by using time series represented as arrays. It should be noted that small range queries are typical in the *MIRABEL* scenario of a real-time smart grid application. This is achieved by keeping the heap organized, only accessing the needed pages and minimizing the disk bandwidth. Figure 8(b) shows a higher speed up by eliminating sorting the time series data. More precisely, we achieve 100 times speed up for small range queries (i.e., selectively of 0.1%) to 1.3 times speed up for almost the entire dataset.

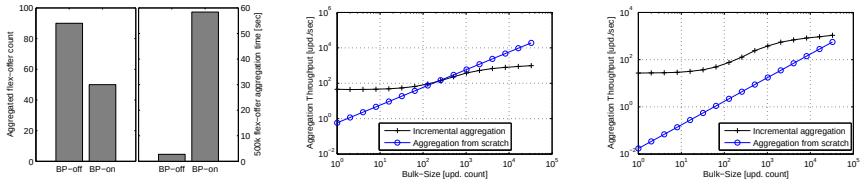
9.2 Aggregation Experiments

We evaluate the throughput of flex-offer updates in the incremental aggregation solution. We use the same experimental setup as in our previous two works [3,9]. Here, the dataset [3] represents synthetically generated energy consumption requests issued for a single day from multiple consumers, and the aggregation



(a) Accessing Timeseries (b) Overall Performance

Fig. 8. Time Series Storage

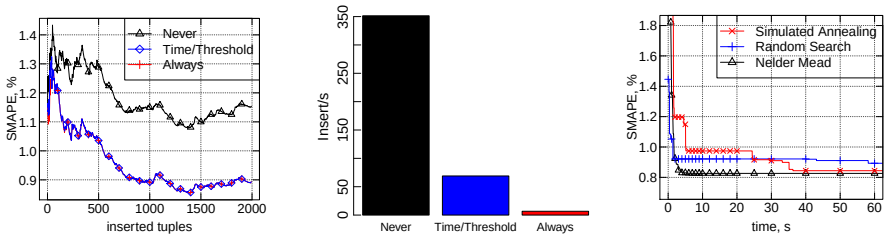


(a) Aggregated flex-offer counts and aggregation times (b) Throughput of updates, BP-off (c) Throughput of updates, BP-on

Fig. 9. Flex-offer Aggregation

parameters are set so that the best scheduling result is obtained. In the experiment we initially aggregate 500000 flex-offers. Then, we continuously feed new flex-offers (for the same 24 hours interval) and remove existing flex-offers while keeping the total number of micro flex-offers equal to 500000. We process such inserts and deletes in bulks. When the bulk of a certain size is formed, we trigger the aggregation. Here, the size of the bulk determines the level of aggregation result up-to-dateness, i.e., small bulks yield up-to-date results while larger bulks yield slightly outdated results. Furthermore, we evaluate our approach with bin-packing (BP) feature on and off. The BP-on means that macro flex-offers are guaranteed to have the time flexibility of no less than 2 hours.

Figure 9(a) visualizes the 500000 flex-offer aggregation performance. When the BP is off, 500000 micro flex-offers are aggregated into 90 macro flex-offers. When the BP is on, the number of macro flex-offer is only 50 as some of the original flex-offers are excluded since they result in macro flex-offers with time flexibility lower than 2 hours (which is not allowed by the aggregate constraint). Times spent aggregating flex-offers with BP-off and BP-on are 3 and 58 seconds, respectively. Figure 9(b) shows the throughput of updates our aggregation solution (with BP-off) can handle when 1) incrementally maintaining macro flex-offers with the bulks of flex-offer updates (line with crosses); 2) aggregating 500000 flex-offers from scratch when a new bulk arrives (line with circles). Similarly, Figure 9(c) shows the throughput when the BP is on. As seen in the



(a) Model evaluation - forecast accuracy (b) Model evaluation - maintenance time (c) Model adaption

Fig. 10. Influence of Maintenance Strategies

figures, the throughput of updates increases when we process updates in larger bulks (applies for both incremental and the from-scratch aggregation, BP-on and -off). For small bulk sizes, the incremental aggregation offers better throughput (even of two orders of magnitude). However, when the bulks are very large (few hundreds of updates in the BP-off case) the better throughput is achieved when aggregating flex-offers from scratch on each bulk of updates. In general, to handle the workload of a particular update rate and to provide as fresh aggregation results as possible, an appropriate bulk size has to be chosen and either incremental or the from-scratch aggregation has to be applied.

9.3 Forecasting Experiments

We evaluate the influence of different model evaluation and adaption strategies (discussed in Section 6) using a publicly available energy production data set [19]. It consists of a single aggregated wind energy time series from 2004 to 2006 in 10 min resolution. The experiment is conducted using Holt-Winters Triple Seasonal Exponential Smoothing, a forecast model tailor-made for the energy domain [12]. To measure the forecast accuracy, we use the symmetric mean absolute percentage error (SMAPE), which is a scale-independent accuracy measure and takes values between 0% and 100%. The following experiments were executed on an IBM Blade (Suse Linux, 64 bit) with two processors and 4 GB RAM.

In a first experiment, we explore the influence of different model evaluation strategies. A first model evaluation strategy *never* adapts the model parameters, while a second strategy *always* adapts the forecast model after each new measurement. Finally, our third strategy triggers model adaption based on a combined time- and threshold-based approach. For this last approach, we set the time limit to 24 hours and the threshold to 2.5%. Figure 10(a) shows the development of the model accuracy for a series of inserts on the time series for each of the three strategies, while Figure 10(b) shows the estimated number of inserts per second we can achieve with the corresponding strategy. First of all, we can observe that model adaption always leads to a lower forecast error compared to no adaption at all and thus is required to achieve the best possible accuracy.

However, if we adapt the model after each new measurement we achieve a much lower throughput than a strategy that never adapts the model parameter. In comparison, our time- and threshold-based approach still leads to a low forecast error but increases the throughput.

In a second experiment, we examine a single model adaption step more closely by comparing the error development of three important parameter estimation strategies (Figure 10(c)). In detail, we measure the accuracy over time of a single adaption step using Simulated Annealing, Random Search and Random Restart Nelder Mead. As it can be seen, all algorithms converge to a result having similar accuracy, where Random Restart Nelder Mead requires much less time than, for example, Simulated Annealing. Thus, the model adaption strategy has a high impact on the required time and needs to be set accordingly.

10 Related Work

In this section, we list other similar smart grid projects, and present work related to the data management components of the *MIRABEL* EDMS.

There are hundreds of ongoing and finished smart grid projects only in Europe [20] with budgets over 3.9 billion euro in total. The most notable are: ADDRESS (25 partners), BeyWatch (8 partners), E-Energy Project "MeRegio" (6 partners), Energy@home (4 partners), INTEGRAL (9 partners), Internet of Energy (40 partners), SmartHouse-SmartGrid (6 partners). The listed ones aim to specify and prototype data exchange infrastructures to demonstrate active customer involvement in the electricity grid operation. The *MIRABEL* project is unique for the "data driven" approach of customer involvement, since the other projects either require end-consumers to negotiate the price and thus actively participate in the electricity market, or are oriented to the distributed energy producers and grid operators. Further, the *MIRABEL* project introduced the flex-offer concept and its precise specification [21] and is now seeking to standardize it [22]. See Section 1 for a discussion of *MIRABEL*'s unique characteristics.

The *MIRABEL* approach exhibits multiple real-time business intelligence requirements and no single system category can cover all of them. Event engines [23], for example, do not support complex analytical queries and advanced analytics (e.g., time series forecasting). Pure data stream management systems (e.g., [24]) can cope with high-rate input streams and rather complex queries; however, ad-hoc data analysis and advanced analytics on historical data are impossible. From the more traditional DBMS side, read-optimized data organization such as column-oriented data management or hybrid OLTP/OLAP solutions [25] provide efficient ad-hoc data analysis but exhibit drawbacks with regard to write-intensive applications. In the data warehouse area, real-time data warehouse approaches try to cope with a continuous stream of write-only updates and read-only queries at the same time [26]. However, the unique concept of flex-offers as well as the requirement of continuous time series forecasting demand an architecture tailor-made for the *MIRABEL* system.

11 Conclusions and Future Work

We have described the real-time business intelligence aspects of the *MIRABEL* system that facilitates a more efficient utilization of renewable energy sources by taking benefit of energy flexibilities. Within *MIRABEL*, real-time advanced analytical queries (including forecasting) have to be supported on a continuous stream of fine-granular time series data and a large number of flex-offers. To cope with these requirements, we introduced a real-time data warehouse solution consisting of the following components. First, the storage management component stores flex-offer and time series data as well as models for historical and future data. With these models, we can provide (1) accurate forecasts of future data, (2) fast approximate results of historical data and (3) increased performance of data loading. Second, the aggregation component efficiently handles a large number of flex-offers by incrementally grouping similar ones with minor flexibility loss. Third, the forecasting component provides accurate forecasts in real-time due to efficient maintenance of energy-domain specific forecast models. Furthermore, the subscription service for flex-offers and forecasts is provided and the standard query processor is extended to support not only exact queries but also approximate queries on past as well as future data. Preliminary experiments on storage management, aggregation, and forecasting components show the applicability and efficiency of the corresponding approaches.

The remaining challenges for the future work are the following: (1) a distribution of the EDMS nodes, e.g., how to ensure consistency and efficiency and lower the amount of communication when propagating the data, (2) a support more types of flexibilities, e.g., price flexibility, and extend data storage management and aggregation components accordingly, (3) a tight integration of forecasting and data loading techniques to enable real-time data warehousing on massive amounts of data in distributed nodes.

With our proposed solution, we take an important step towards the integration of larger amounts of distributed generated RES into the electricity grid.

Acknowledgment. The work presented in this paper has been carried out in the MIRABEL project funded by the EU under the grant agreement number 248195.

References

1. *The MIRABEL Project*, 2012. `www.mirabel-project.eu/`.
2. ETSO, "The harmonized electricity market role model." `https://www.entsoe.eu/fileadmin/user_upload/edi/library/role/role-model-v2009-01.pdf`, 2009.
3. T. Tušar, L. Šikšnys, T. B. Pedersen, E. Dovgan, and B. Filipič, "Using aggregation to improve the scheduling of flexible energy offers," in *Proc. of BIOMA*, 2012.
4. L. Šikšnys, C. Thomsen, and T. B. Pedersen, "MIRABEL DW: Managing Complex Energy Data in a Smart Grid," in *Proc. of DaWaK*, 2012.
5. A. Thiagarajan and S. Madden, "Querying continuous functions in a database system," in *Proc. of SIGMOD*, pp. 791–804, 2008.

6. H. Shatkay and S. B. Zdonik, "Approximate queries and representations for large data sequences," in *Proc. of ICDE*, pp. 536–545, 1996.
7. M. E. Khalefa, U. Fischer, and T. B. Pedersen, "Model-based Integration of Past & Future in TimeTravel (demo)," in *PVLDB*, 2012.
8. C. Thomsen, T. B. Pedersen, and W. Lehner, "RiTE: Providing On-Demand Data for Right-Time Data Warehousing," in *Proc. of ICDE*, pp. 456–465, 2008.
9. L. Šikšnys, M. E. Khalefa, and T. B. Pedersen, "Aggregating and disaggregating flexibility objects," in *Proc. of SSDBM*, 2012.
10. M. R. Garey and D. S. Johnson, *Computers and Intractability; A Guide to the Theory of NP-Completeness*. New York, NY, USA: W. H. Freeman & Co., 1990.
11. M. Boehm, L. Dannecker, A. Doms, E. Dovgan, B. Filipic, U. Fischer, W. Lehner, T. B. Pedersen, Y. Pitarch, L. Siksnys, and T. Tusar, "Data management in the mirabel smart grid system," in *Proc. of EnDM*, 2012.
12. J. W. Taylor, "Triple Seasonal Methods for Short-Term Electricity Demand Forecasting," *European Journal of Operational Research*, 2009.
13. R. Ramanathan, R. Engle, C. W. J. Granger, F. Vahid-Araghi, and C. Brace, "Short-run Forecasts of Electricity Loads and Peaks," *International Journal of Forecasting*, vol. 13, no. 2, pp. 161–174, 1997.
14. L. Dannecker, M. Boehm, W. Lehner, and G. Hackenbroich, "Forcasting evolving time series of energy demand and supply," in *Proc. of ADBIS*, 2011.
15. L. Dannecker, R. Schulze, M. Boehm, W. Lehner, and G. Hackenbroich, "Context-aware parameter estimation for forecast models in the energy domain," in *Proc. of SSDBM*, 2011.
16. L. Dannecker, M. Boehm, W. Lehner, and G. Hackenbroich, "Partitioning and multi-core parallelization of multi-equation forecast models," in *Proc. of SSDBM*, 2012.
17. U. Fischer, M. Boehm, and W. Lehner, "Offline design tuning for hierarchies of forecast models," in *Proc. of BTW*, 2011.
18. U. Fischer, M. Boehm, W. Lehner, and T. B. Pedersen, "Optimizing notifications of subscription-based forecast queries," in *Proc. of SSDBM*, 2012.
19. NREL, *Wind Integration Datasets*, 2012. `http://www.nrel.gov/wind/integrationdatasets/`.
20. *JRC study on Smart Grid projects in Europe*, 2011. `http://ses.jrc.ec.europa.eu/index.php?option=com_content&view=article&id=93&Itemid=137`.
21. M. Konsman and F.-J. Rumph, "D2.3. Final data model, specification of request and negotiation messages and contracts," 2011. `http://wwwdb.inf.tu-dresden.de/miracle/publications/D2.3.pdf`.
22. J. Verhoosel, D. Rothengatter, F.-J. Rumph, and M. Konsman, "D7.5. MIRABEL-ONE: Initial draft of the MIRABEL Standard," 2011. `http://wwwdb.inf.tu-dresden.de/miracle/publications/D7.5.pdf`.
23. E. Wu, Y. Diao, and S. Rizvi, "High-performance complex event processing over streams," in *Proc. of SIGMOD*, 2006.
24. D. Abadi, D. Carney, U. Cetintemel, M. Cherniack, C. Convey, C. Erwin, E. Galvez, M. Hatoun, J. h. Hwang, A. Maskey, A. Rasin, A. Singer, M. Stonebraker, N. Tatbul, Y. Xing, R. Yan, and S. Zdonik, "Aurora: A data stream management system," in *Proc. of SIGMOD*, 2003.
25. A. Kemper and T. Neumann, "Hyper: A hybrid oltp & olap main memory database system based on virtual memory snapshots," in *Proc. of ICDE*, 2011.
26. M. Thiele and W. Lehner, "Evaluation of load scheduling strategies for real-time data warehouse environments," in *Proc. of BIRTE*, 2009.